Structural Induction and the λ-Calculus (Algebra, Logic and Geometry in Informatics)

AUTHOR(S):

Vestergaard, Rene

CITATION:

Vestergaard, Rene. Structural Induction and the λ-Calculus (Algebra, Logic and Geometry in Informatics). 数理解析研究所講究録 2003, 1318: 30-45

ISSUE DATE:

2003-05

URL:

http://hdl.handle.net/2433/43027

数理解析研究所講究録 1318 巻 2003 年 30-45

Structural Induction and the λ-Calculus

René Vestergaard*

School of Information Science
Japan Advanced Institute of Science and Technology

Abstract. We consider formal provability with structural induction and related proof principles in the λ-calculus presented with first-order abstract syntax over one-sorted variable names. As well as summarising and elaborating on earlier, formally verified proofs (in Isabelle/HOL) of the relative renaming-freeness of β-residual theory and β-confluence, we also present proofs of η-confluence, βη-confluence, the strong weakly-finite β-development (aka residual-completion) property, residual β-confluence, η-over-β-postponement, and notably β-standardisation. In the latter case, the known proofs fail in instructive ways. Interestingly, our uniform proof methodology, which has relevance beyond the λ-calculus, properly contains pen-and-paper proof practices in a precise sense. The proof methodology also makes precise what is the full algebraic proof burden of the considered results, which we, moreover, appear to be the first to resolve.

1 Introduction

The use of structural induction and related proof principles for simple syntax (i.e., first-order abstract syntax over one-sorted variable names) is a long-standing and widely-used practice in the programming-language theory community. Unfortunately, at a first, closer inspection it seems that the practice is not formally justifiable because of a need to avoid undue variable capture when performing substitution, thus breaking the syntactic equality underlying structural induction, etc.. Even more worrying is the fact that, in spite of substantial efforts in the mechanised theorem-proving community, no formal proof developments (prior to what we report on here) have been able to overcome the problems that are encountered with substitution and go on to successfully employ the proof principles in question. Indeed, and starting with de Bruijn [6], it has become an active research area to define, formalise, and automate alternative syntactic frameworks that, on the one hand, preserve as much of the inherent naturality of simple syntax as possible. At the same time, they are customised to provide suitable induction and recursion principles for any considered language [6–10, 12, 17, 21]. However, by changing the underlying syntactic framework, the algebraic meaning of, e.g., a diamond property also changes, which means that, e.g., confluence as proved and as defined no longer coincide, cf. Lemma 18 and [25].

In the recognition that the above is both unfortunate as far as the formal status of the existing informal literature is concerned and unsatisfactory from a mathematical perspective, we pursue the naive approach in this article (while incorporating the relevant aspects of [24, 25]). In particular, we show that it is, indeed, possible to base formal proofs on first-order abstract syntax over one-sorted variable names and hope to convince the reader that, while the technical gap between pen-and-paper and formal proofs is rather large, the conceptual gap is somewhat smaller. Furthermore, we hope that the comprehensive range of applications of the proof methodology that we present here will establish its wider relevance.

1.1 Syntax of the λ-Calculus

The λ-calculus is intended to capture the concept of a *function*. It does so, first of all, by providing syntax that can be used to express function application and definition:

$$e ::= x \mid e_1 e_2 \mid \lambda x.e$$

The above, informal syntax says that a λ-term, e, is defined inductively as either a variable name, as an application of one term to another, or as a λ-, or functional, abstraction of a variable name over a term. The variable names, x, are typically taken to be, or range over, words over the Latin alphabet. In Section 2, we will review the exact requirements to variable names in an abstract sense. Being based on a simple, inductive definition, λ-terms also come equipped with a range of primitive proof principles [1, 3].

Syntactic Equality As a λ-term, e, is finite and consists of variable names, the obvious variable-name equality, $=_{\mathcal{VN}}$, which exists at least in the case of words over the Latin alphabet, canonically extends to all λ-terms:

$$\frac{x =_{\mathcal{VN}} y}{x =_{\Lambda^{\mathrm{var}}} y} \qquad \frac{e_1 =_{\Lambda^{\mathrm{var}}} e_1' \quad e_2 =_{\Lambda^{\mathrm{var}}} e_2'}{e_1 e_2 =_{\Lambda^{\mathrm{var}}} e_1' e_2'} \qquad \frac{x =_{\mathcal{VN}} y \quad e =_{\Lambda^{\mathrm{var}}} e'}{\lambda x.e =_{\Lambda^{\mathrm{var}}} \lambda y.e'}$$

Structural Induction In order to prove properties about the set of λ-terms, we can proceed by means of *structural induction*, mimicking the inductive definition of the terms:

$$\frac{\forall x.P(x) \;\; \forall e_1, e_2.P(e_1) \wedge P(e_2) \Rightarrow P(e_1 e_2) \;\; \forall x, e.P(e) \Rightarrow P(\lambda x.e)}{\forall e.P(e)}$$

* vester@jaist.ac.jp, http://www.jaist.ac.jp/~vester

$$y[x := e]_{\mathrm{Cu}} = \begin{cases} e & \text{if } x = y \\ y & \text{otherwise} \end{cases}$$

$$(e_1 e_2)[x := e]_{\mathrm{Cu}} = e_1[x := e]_{\mathrm{Cu}} e_2[x := e]_{\mathrm{Cu}}$$

$$(\lambda y.e_0)[x := e]_{\mathrm{Cu}} = \begin{cases} \lambda y.e_0 & \text{if } x = y \\ \lambda y.e_0[x := e]_{\mathrm{Cu}} & \text{if } x \neq y \wedge (y \notin \mathrm{FV}(e) \vee x \notin \mathrm{FV}(e_0)) \\ \lambda z.e_0[y := z]_{\mathrm{Cu}}[x := e]_{\mathrm{Cu}} & \text{o/w; } \textit{first } z \notin \{x\} \cup \mathrm{FV}(e) \cup \mathrm{FV}(e_0) \end{cases}$$

Fig. 1. Curry-style capture-avoiding substitution

$$\frac{}{x[x := e]_{\mathrm{Cu}} = e} \qquad \frac{x \neq y}{y[x := e]_{\mathrm{Cu}} = y} \qquad \frac{e_1[x := e]_{\mathrm{Cu}} = e_1' \quad e_2[x := e]_{\mathrm{Cu}} = e_2'}{(e_1 e_2)[x := e]_{\mathrm{Cu}} = e_1' e_2'}$$

$$\frac{}{(\lambda x.e_0)[x := e]_{\mathrm{Cu}} = \lambda x.e_0} \qquad \frac{x \neq y \quad (y \notin \mathrm{FV}(e) \vee x \notin \mathrm{FV}(e_0')) \quad e_0[x := e]_{\mathrm{Cu}} = e_0'}{(\lambda y.e_0)[x := e]_{\mathrm{Cu}} = \lambda y.e_0'}$$

$$\frac{x \neq y \quad y \in \mathrm{FV}(e) \quad x \in \mathrm{FV}(e_0) \quad z = \mathrm{Fresh}((e_0 e)x) \quad e_0[y := z]_{\mathrm{Cu}} = e_0' \quad e_0'[x := e]_{\mathrm{Cu}} = e_0''}{(\lambda y.e_0)[x := e]_{\mathrm{Cu}} = \lambda z.e_0''}$$

Fig. 2. Curry-style substitution (re-)defined inductively

Structural Case-Splitting As each syntax constructor of the λ-calculus is unique, we see that it is possible to case-split on terms — with E_i in some suitable meta-language:

$$\begin{array}{rl} \text{case } e \text{ of} & x \Rightarrow E_1(x) \\ \mid & e_1 e_2 \Rightarrow E_2(e_1, e_2) \\ \mid & \lambda x.e_0 \Rightarrow E_3(x, e_0) \end{array}$$

Structural Recursion Based on case-splitting and well-foundedness of terms, we can even define functions on λ-terms by means of structural recursion, i.e., by making recursive calls only on the sub-terms of a given constructor:

$$\begin{aligned} f(x) &= E_1(x) \\ f(e_1 e_2) &= E_2(f(e_1), f(e_2)) \\ f(\lambda x.e) &= E_3(x, f(e)) \end{aligned}$$

The above implies that f is well-defined: it is computable by virtue of well-foundedness of terms and total because the definition case-splits exhaustively on λ-terms. As an example application, we define the function that computes the *free variables* in a term, i.e., the variable names that do not occur inside a λ-abstraction of themselves.

Definition 1

$$\begin{aligned} \mathrm{FV}(y) &= \{y\} \\ \mathrm{FV}(e_1 e_2) &= \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2) \\ \mathrm{FV}(\lambda y.e) &= \mathrm{FV}(e) \setminus \{y\} \end{aligned}$$

Proposition 2 $\mathrm{FV}(-)$ *is a total, computable function.*

1.2 Reduction and Substitution

In order to have λ-abstractions act as functions and not to have too many, e.g., identity functions, amongst other things, we are typically interested in the following relations that can be applied anywhere in a term — their precise form is due to Curry [4].

1. $(\lambda x.e)e' \dashrightarrow_{\beta^{\mathrm{Cu}}} e[x := e']_{\mathrm{Cu}}$
2. $\lambda y.e[x := y]_{\mathrm{Cu}} \dashrightarrow_{\alpha^{\mathrm{Cu}}} \lambda x.e$, if $y \notin \mathrm{FV}(e)$

Our interest in 2., above is the equivalence relation it induces. We denote it by $=\!=_\alpha$, cf. Appendix B, and we will eventually factor it out, as is standard.

Variable Capture In his seminal formalist presentation of the λ-calculus [4], Curry defines the above *substitution* operator, $-[- := -]_{\mathrm{Cu}}$, essentially as in Figure 1. The last clause is the interesting one. It renames the considered y into the first z that has not been used already.[1] Consider, for example, the substitution of x for z in the two terms $\lambda x.z$ and $\lambda y.z$. Both terms-as-functions discard their argument. If we simply replace the z in the terms with x, the latter would still discard its argument but the former would become the identity function and this discrepancy would lead to inconsistencies.

Well-Definedness Of formalist relevance, we remark that Curry-style substitution is not well-defined by construction as the definition does not employ structural

[1] While the notion "the first z" is trivially well-defined in the present case, the issue is a bit more subtle in a wider context, as we shall see in Section 2.

recursion. The offender is the last clause that applies $-[x := e]$ to a term, $e_0[y := z]$, which is not a subterm of $\lambda y.e_0$ in general. It can be observed that while $e_0[y := z]$ is not a sub-term of $\lambda y.e_0$, it will have the same size as e_0 and we can thus establish the well-formedness of $-[- := -]_{\mathrm{Cu}}$ by external means. Alternatively, we can introduce a more advanced, *parallel substitution* operator [22]. However, as we eventually will distance ourselves from the use of renaming in substitution, we will do neither but instead refer to Section 2.3 for an alternative derivation of Curry-style substitution.

Variable-Name Indeterminacy Having initially committed ourselves to using renaming in substitution, a range of problems are brought down on us. Hindley [11] observed, for example, that it becomes impossible to predict the variable name used for a given abstraction after reducing, thus putting, e.g., confluence out of reach:

$$(\lambda x.(\lambda y.\lambda x.xy)x)y \begin{array}{l} \xrightarrow[\beta^{\mathrm{Cu}}]{} (\lambda y.\lambda x.xy)y \xrightarrow[\beta^{\mathrm{Cu}}]{} \lambda x.xy \\ \xrightarrow[\beta^{\mathrm{Cu}}]{} (\lambda x.\lambda z.zx)y \xrightarrow[\beta^{\mathrm{Cu}}]{} \lambda z.zy \end{array}$$

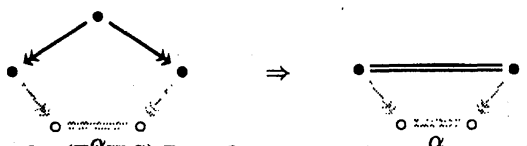
In the lower branch, the innermost x-abstraction must be renamed to a z-abstraction, while the upper branch never encounters the variable-name clash. Hindley proceeded to define a β-relation on α-equivalence classes that overcomes the above indeterminacy by factoring it out:

$$\lfloor e \rfloor =^{\mathrm{def}} \{e' \mid e =\!=_\alpha e'\}$$
$$\lfloor e_1 \rfloor \to_{\beta^{\mathrm{Hi}}} \lfloor e_2 \rfloor =^{\mathrm{def}} \exists e_1' \in \lfloor e_1 \rfloor, e_2' \in \lfloor e_2 \rfloor . e_1' \dashrightarrow_{\beta^{\mathrm{Cu}}} e_2'$$

No relevant proof principles are introduced by this and the approach can not be used in a formal setting as it stands.

Broken Induction Steps Instead of factoring out α-equivalence altogether, one could attempt to reason up to post-fixed name unification. Unfortunately, this would lead to a range of unusual situations as far as subsequent uses of abstract rewriting is concerned. An example is the following attempted adaptation of the well-known equivalence between confluence and the Church-Rosser property. Please refer to Appendix A for a precise definition of our diagram notation.

Non-Lemma 3



Proof (FAILS) By reflexive, transitive, symmetric induction in $=$.

Base, Reflexive, Symmetric Cases: Simple.

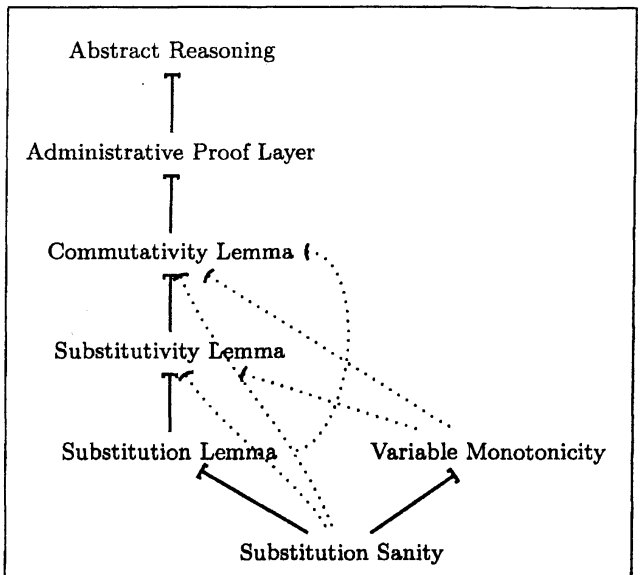
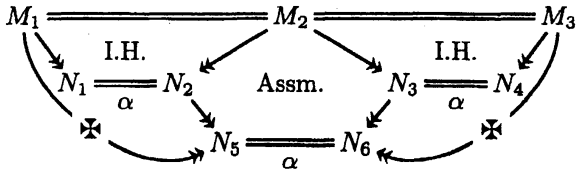


Fig. 3. The proof-layer hierarchy for primitive equational reasoning about the λ-calculus as simple syntax

Transitive Case: Breaks down.



Broken α-Equality in Sub-Terms Having failed in our attempts to control limited use of α-equivalence, one might think that the syntactic version of Hindley's approach, cf. Section 1.2, could work: that it is possible to state all properties about terms up to $=\!=_\alpha$ rather than the primitive $=_{\Lambda^{\mathrm{var}}}$.

Lemma 4 (Simplified Substitution modulo α)

$$e_1 =\!=_\alpha e_2 \wedge x \neq y_i \wedge y_1 \neq y_2$$
$$\Downarrow$$
$$e_1[x_1 := y_1]_{\mathrm{Cu}}[x_2 := y_2]_{\mathrm{Cu}} =\!=_\alpha e_2[x_2 := y_2]_{\mathrm{Cu}}[x_1 := y_1]_{\mathrm{Cu}}$$

Proof (FAILS) By structural induction in e_1.

Most Cases: Trivial.

Last Abstraction Case (simplified): Breaks down.

$$\begin{aligned} & (\lambda y_1.e)[x_1 := y_1]_{\mathrm{Cu}}[x_2 := y_2] \\ &= \lambda z.e'[x_1 := y_1]_{\mathrm{Cu}}[x_2 := y_2]_{\mathrm{Cu}} \\ &=\!=_\alpha^{\not\Downarrow} \lambda z.e'[x_2 := y_2]_{\mathrm{Cu}}[x_1 := y_1]_{\mathrm{Cu}} \\ &= (\lambda z.e')[x_2 := y_2]_{\mathrm{Cu}}[x_1 := y_1]_{\mathrm{Cu}} \end{aligned}$$

The problem above is that e and e' are *not* actually α-equivalent, even if $\lambda y_1.e$ and $\lambda z.e'$ are, and the $=\!=_\alpha$-step can thus *not* be substantiated by the induction hypothesis. Consider, e.g., e as y_1 and e' as z. The above result is certainly correct but, unfortunately, not provable with the tools we have at our disposal at the moment.

1.3 This article

The results we are dealing with are mostly well-known and have been addressed in several contexts. Indeed, a number of truly beautiful and concise informal proofs exist; see, in particular, Takahashi [23], whom we owe a great debt. This article, therefore, spends little energy on those parts of the proofs and focuses instead on what it takes to formalise them. There are two key issues: (i) the syntactic properties that can actually be established up to $=_{\Lambda^{\mathrm{var}}}$ (as opposed to $==_\alpha$, which we have seen to be highly problematic) and (ii) how to generalise these to the algebraic properties we are seeking. The full typeset proofs (roughly 100 pages for the proofs alone) are available from our homepage.

In general, our proofs follow the structure that we present in Figure 3. It is based on nested inductions. The full-coloured arrows mean "is the key lemma for", while the others mean "is used to substantiate side-conditions on lemma applications". The first issue above, (i), is expressed in the addition of the "Variable Monotinicity" proof layer in Figure 3. The second issue, (ii), is entirely accounted for in the "Administrative Proof Layer" in Figure 3.

The proofs underpinning Sections 3 and 4.1 have been verified in full in Isabelle/HOL (at least in the case of one of the alternatives they present) [24, 25]. By the nature of Figure 3, this means that substantial parts of the other proofs essentially have been verified as well.

Apart from the various technical sections in the body of this paper, the appendix section contains an explanation of our diagram notation (Appendix A) and our other notation (Appendix B) as well as some well-known rewriting results that we use (Appendix C).

2 The λ^{var}-Calculus

Having seen that the standard presentations of the λ-calculus lead to formalist problems, we will now give an alternative presentation that overcomes them. The different presentations differ only in how they lend themselves to provability. Their equational properties are equivalent.

2.1 Formal Syntax

We use e's to range over the inductively built-up set of λ-terms. The variable names, $\mathcal{VN}$, are generic but must meet certain minimal requirements.

Definition 5 $\Lambda^{\mathrm{var}} ::= \mathcal{VN} \mid \Lambda^{\mathrm{var}}\Lambda^{\mathrm{var}} \mid \lambda\mathcal{VN}.\Lambda^{\mathrm{var}}$

Assertion 6 *$\mathcal{VN}$ is a single-sorted set of objects, aka* variable names.

Assertion 7 *$\mathcal{VN}$-equality, $=_{\mathcal{VN}}$, is decidable.*

$$\begin{aligned}
\mathrm{BV}(y) &= \emptyset \\
\mathrm{BV}(e_1 e_2) &= \mathrm{BV}(e_1) \cup \mathrm{BV}(e_2) \\
\mathrm{BV}(\lambda y.e) &= \{y\} \cup \mathrm{BV}(e)
\end{aligned}$$

$$\begin{aligned}
\mathrm{Capt}_x(y) &= \emptyset \\
\mathrm{Capt}_x(e_1 e_2) &= \mathrm{Capt}_x(e_1) \cup \mathrm{Capt}_x(e_2) \\
\mathrm{Capt}_x(\lambda y.e) &= \begin{cases} \{y\} \cup \mathrm{Capt}_x(e) & \text{if } x \in \mathrm{FV}(\lambda y.e) \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned}$$

Fig. 4. Bound and capturing variable names

Assertion 8 *There exists a total, computable function,* $\mathrm{Fresh}(-) : \Lambda^{\mathrm{var}} \longrightarrow \mathcal{VN}$, *such that:*[2]

$$\mathrm{Fresh}(e) \notin \mathrm{FV}(e) \cup \mathrm{BV}(e)$$

The last assertion trivially implies that $\mathcal{VN}$ is infinite.[3]

We shall use x's, y's, and z's as meta-variables of $\mathcal{VN}$ and, by a slight abuse of notation, also as actual variable names in terms. We will suppress the $\mathcal{VN}$ suffix on variable-name equality and merely write, e.g., $x = y$.

2.2 Orthonormal Reduction

The key technicality to prevent implicit renaming is our use of a predicate, $\mathrm{Capt}_x(e_1) \cap \mathrm{FV}(e_2) = \emptyset$, cf. Figure 4, which guarantees that no capture takes place in the substitution: $e_1[x := e_2]$. It coincides with the notion of *not free for.*

Definition 9 (The λ^{var}-Calculus) *The terms of the λ^{var}-calculus are Λ^{var}, cf. Definition 5. The (indexed) α-, β-, and η-reduction relations of λ^{var}: $\overline{\dashrightarrow}_{i\alpha}$, $\dashrightarrow_\beta$, and $\dashrightarrow_\eta$ are given inductively in Figure 5. The plain α-relation is:*

$$e \dashrightarrow_\alpha e' \Leftrightarrow^{\mathrm{def}} \exists y.e \overset{y}{\overline{\dashrightarrow}}_{i\alpha} e'$$

Unlike the situation with Curry-style substitution, we see that our notion of substitution is defined by structural recursion and, hence, is well-defined by construction.

Proposition 10 *$-[x := e]$ is a total, computable function.*

[2] For the definition of $\mathrm{BV}(-)$, see Figure 4.

[3] In the setting of Nominal Logic [19], the assertion also validates the axiom of choice, which is known to be provably inconsistent with the Fraenkel-Mostowski set theory that underpins Nominal Logic. Nominal Logic instead guarantees the existence of *some* fresh variable name, which by design can be *any* variable name except for a finite number. More work needs to be done to clarify the correspondence between simple syntax and syntax based on Nominal Logic.

$$y[x := e] = \begin{cases} e \text{ if } x = y \\ y \text{ otherwise} \end{cases}$$
$$(e_1 e_2)[x := e] = e_1[x := e] e_2[x := e]$$
$$(\lambda y.e_0)[x := e] = \begin{cases} \lambda y.e_0[x := e] \text{ if } x \neq y \wedge y \notin \mathrm{FV}(e) \\ \lambda y.e_0 \quad \text{otherwise} \end{cases}$$

$$\frac{y \notin \mathrm{Capt}_x(e) \cup \mathrm{FV}(e)}{\lambda x.e \xrightarrow{y}_{i\alpha} \lambda y.e[x := y]}\ (\alpha) \qquad \frac{e \xrightarrow{y}_{i\alpha} e'}{\lambda x.e \xrightarrow{y}_{i\alpha} \lambda x.e'} \qquad \frac{e_1 \xrightarrow{y}_{i\alpha} e_1'}{e_1 e_2 \xrightarrow{y}_{i\alpha} e_1' e_2} \qquad \frac{e_2 \xrightarrow{y}_{i\alpha} e_2'}{e_1 e_2 \xrightarrow{y}_{i\alpha} e_1 e_2'}$$

$$\frac{\mathrm{Capt}_x(e_1) \cap \mathrm{FV}(e_2) = \emptyset}{(\lambda x.e_1) e_2 \dashrightarrow_\beta e_1[x := e_2]}\ (\beta) \qquad \frac{e \dashrightarrow_\beta e'}{\lambda x.e \dashrightarrow_\beta \lambda x.e'} \qquad \frac{e_1 \dashrightarrow_\beta e_1'}{e_1 e_2 \dashrightarrow_\beta e_1' e_2} \qquad \frac{e_2 \dashrightarrow_\beta e_2'}{e_1 e_2 \dashrightarrow_\beta e_1 e_2'}$$

$$\frac{x \notin \mathrm{FV}(e) = \emptyset}{\lambda x.ex \dashrightarrow_\eta e}\ (\eta) \qquad \frac{e \dashrightarrow_\eta e'}{\lambda x.e \dashrightarrow_\eta \lambda x.e'} \qquad \frac{e_1 \dashrightarrow_\eta e_1'}{e_1 e_2 \dashrightarrow_\eta e_1' e_2} \qquad \frac{e_2 \dashrightarrow_\eta e_2'}{e_1 e_2 \dashrightarrow_\eta e_1 e_2'}$$

Fig. 5. Renaming-free substitution, $-[- := -]$, defined recursively, and α-, β-, η-reduction defined inductively over Λ^{var}

The β- and η-relations we have presented above do not incur any renaming that could have been performed in a stand-alone fashion by the α-relation, thus making them *orthogonal*. The *normality* part of our informal orthonormality principle is established by the following property, symmetry of $\dashrightarrow_\alpha$, which implies that the α-relation itself is renaming-free.

Lemma 11 

2.3 Curry's λ-Calculus Decomposed

In order to assure ourselves that the λ^{var}-calculus is indeed the right calculus and partly to test the usefulness of the associated primitive proof principles, we now show how to derive Curry's presentation from ours. First, we show that as far as our use of substitution is concerned, $-[- := -]$ coincides with $-[- := -]_{\mathrm{Cu}}$.

Proposition 12

$$\mathrm{Capt}_x(e_a) \cap \mathrm{FV}(e) = \emptyset$$
$$\Downarrow$$
$$e_a[x := e] = e_a[x := e]_{\mathrm{Cu}}$$

Proof A straightforward structural induction in e_a. □

What might not be obvious is that Curry-style substitution can be shown to decompose into the λ^{var}-calculus. In contrast to the structurally flawed Figure 1, Figure 2 introduces a primitively-defined, 4-ary relation that *is* Curry-style substitution, albeit with no claim of well-definedness.

Lemma 13

$$e_a[x := e]_{\mathrm{Cu}} = e_a'$$
$$\Downarrow$$
$$\exists! e_b . e_a \dashrightarrow\!\!\!\!\!\rightarrow_\alpha e_b \wedge e_b[x := e] = e_a'$$

Proof By rule induction in Curry-style substitution-as-a-relation, cf. Figure 2. Uniqueness of e_b is guaranteed by the functionality of Fresh$(-)$. □

We stress that the above property is not provable by structural induction in e_a and that it ensures that Curry-style substitution is, indeed, well-defined and functional.

Lemma 14 *For any x and e, $-[x := e]_{\mathrm{Cu}} = -$ is a total, computable function of the first, open argument onto the second, open argument.*

Lemma 13 also establishes the decomposition of Curry's calculus as a whole into the λ^{var}-calculus.

Lemma 15 $\dashrightarrow_\alpha \subseteq (\dashrightarrow_{\alpha^{\mathrm{Cu}}})^{-1} \subseteq \dashrightarrow\!\!\!\!\!\rightarrow_\alpha$

Lemma 16 $\dashrightarrow_\beta \subseteq \dashrightarrow_{\beta^{\mathrm{Cu}}} \subseteq \dashrightarrow\!\!\!\!\!\rightarrow_\alpha ; \dashrightarrow_\beta$

2.4 The Real λ-Calculus

As suggested previously, the actual calculus we are interested in is the α-collapse of λ^{var}. Algebraically speaking, this means that we want to consider the following structure, cf. Hindley's presentation, Section 1.2.

Definition 17 (The Real λ-Calculus)

$$- \Lambda =^{def} \Lambda^{\mathrm{var}} / =\!=_\alpha$$

$$\begin{aligned}\mathrm{UB}(x) &= \mathrm{True}\\ \mathrm{UB}(e_1 e_2) &= \mathrm{UB}(e_1) \wedge \mathrm{UB}(e_2) \wedge (\mathrm{BV}(e_1) \cap \mathrm{BV}(e_2) = \emptyset)\\ \mathrm{UB}(\lambda x.e) &= \mathrm{UB}(e) \wedge x \notin \mathrm{BV}(e)\end{aligned}$$

Fig. 6. The *uniquely bound* Λ^{var}-predicate

$$\begin{aligned}&- \lfloor - \rfloor :^{def} \Lambda^{\mathrm{var}} \longrightarrow \Lambda\\ &\qquad\qquad e \mapsto \{e' \mid e =\!=_\alpha e'\}\\ &- \lfloor e_1 \rfloor \longrightarrow_\beta \lfloor e_2 \rfloor \Leftrightarrow^{def} e_1 =\!=_\alpha ; \dashrightarrow_\beta ; =\!=_\alpha e_2\\ &- \lfloor e_1 \rfloor \longrightarrow_\eta \lfloor e_2 \rfloor \Leftrightarrow^{def} e_1 =\!=_\alpha ; \dashrightarrow_\eta ; =\!=_\alpha e_2\end{aligned}$$

It can be shown (without too much trouble) that Curry's, Hindley's, and our relations all are *pointwise* identical, cf. [25]. For now, we merely present the part of that result that pertains to the current set-up.

Lemma 18 *For* $\mathrm{X} \in \{\beta, \eta, \beta\eta\}$ *(any* X*, in fact), we have:*

$$\lfloor e \rfloor \longrightarrow\!\!\!\!\!\rightarrow_{\mathrm{X}} \lfloor e' \rfloor \Leftrightarrow e \dashrightarrow\!\!\!\!\!\rightarrow_{\alpha \mathrm{X}} e'$$

Proof By definition of the real relations and reflexive, transitive closure, we immediately see that

$$\lfloor e \rfloor \longrightarrow\!\!\!\!\!\rightarrow_{\mathrm{X}} \lfloor e' \rfloor \Leftrightarrow e\ (=\!=_\alpha ; \dashrightarrow_{\mathrm{X}} ; =\!=_\alpha)^\star\ e' \vee e =\!=_\alpha e'$$

The result thus follows directly from Lemma 11. □

3 Residual Theory

This section shows that residual theory, i.e., the exclusive contraction of pre-existing, or marked, redexes, provides a nice setting for quantifying the "computing power" of the renaming-free β-relation. We use t_i's as meta-variables over the marked terms and we allow ourselves to use Λ^{var}-concepts for the marked terms with only implicit coercions; in particular, we assume there is an $\alpha^{@}$-relation that can rename all (not just marked) abstractions.

Definition 19 (The Marked λ^{var}-Calculus)

$$\Lambda^{\mathrm{var}}_{@} ::= x \mid \Lambda^{\mathrm{var}}_{@} \Lambda^{\mathrm{var}}_{@} \mid \lambda \mathcal{VN}.\Lambda^{\mathrm{var}}_{@} \mid (\lambda \mathcal{VN}.\Lambda^{\mathrm{var}}_{@}) @ \Lambda^{\mathrm{var}}_{@}$$

$\dashrightarrow_{\beta@}$ *is like* $\dashrightarrow_\beta$ *except only marked redexes,* $(\lambda x.t_1) @ t_2$*, may be contracted (provided* $\mathrm{Capt}_x(t_1) \cap \mathrm{FV}(t_2) = \emptyset$*). We further define a* residual-completion *relation,* $\dashrightarrow\!\!\!\!\!\rightarrow_{\beta@}$*, by induction over terms that attempts to contract all (marked) redexes in one step, starting from within.*[4]

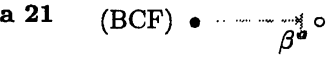
[4] The relation corresponds closely to the parallel β-relation of Figure 7.

To address any inherent requirements for renaming in the λ-calculus, we introduce a formal notion called *Barendregt Conventional Form* (BCF),[5] which, as it turns out, provides a rational reconstruction of the usual (informal) Barendregt Variable Convention [2], cf. [25]. BCFs are terms where all variable names are different.

Definition 20 *Cf. Figures 4 and 6:*

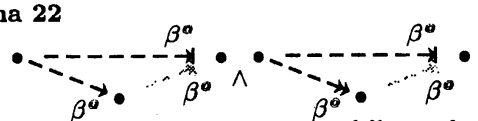
$$\mathrm{BCF}(e) = \mathrm{UB}(e) \wedge (\mathrm{BV}(e) \cap \mathrm{FV}(e) = \emptyset)$$

As a first approximation to renaming-freeness, we note that it is a straightforward proof that BCFs residually completes, i.e., that all marked redexes in a BCF can be contracted from within without causing variable clashes.

Lemma 21 (BCF) 

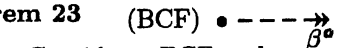
We also show that the residual-completion relation is functional on the full β-residual theory of a term, i.e., that residual completion always catches up with itself.

Lemma 22



Proof The right-most conjunct follows from the left-most by a simple reflexive, transitive induction in which the latter constitutes the base case. The left-most conjunct follows by a rule induction in $\dashrightarrow\!\!\!\!\!\rightarrow_{\beta@}$ for which it is paramount that redexes are enabled if $\mathrm{Capt}_x(-) \cap \mathrm{FV}(-) = \emptyset$ rather than only if $\mathrm{BV}(-) \cap \mathrm{FV}(-) = \emptyset$. Other than that, the proof is mostly straightforward, albeit big. □

The above property asserts that *when* residual completion exists, the considered divergence can be resolved as shown. The property allows us to prove that β-residual theory is renaming-free up to BCF-initiality, i.e., that no redexes are blocked by their side-condition.

Theorem 23 (BCF) 

Proof Consider a BCF and a $\dashrightarrow\!\!\!\!\!\rightarrow_{\beta@}$-reduction of it. By Lemma 21, the considered BCF also residually completes and, by Lemma 22, the thus-created divergence can be resolved by a trailing residual completion. □

A subtle point of interest is that the above proof, in fact, shows that the β-residual theory of any term that residually-completes, i.e., is renaming-free if contracted from within, is renaming-free in general.

[5] The term was suggested to us by Randy Pollack.

$$\frac{}{x \twoheadrightarrow_\beta x} \qquad \frac{e \twoheadrightarrow_\beta e'}{\lambda x.e \twoheadrightarrow_\beta \lambda x.e'} \qquad \frac{e_1 \twoheadrightarrow_\beta e_1' \quad e_2 \twoheadrightarrow_\beta e_2'}{e_1 e_2 \twoheadrightarrow_\beta e_1' e_2'}$$

$$\frac{e_1 \twoheadrightarrow_\beta e_1' \quad e_2 \twoheadrightarrow_\beta e_2' \quad \mathrm{FV}(e_2') \cap \mathrm{Capt}_x(e_1') = \emptyset}{(\lambda x.e_1)e_2 \twoheadrightarrow_\beta e_1'[x := e_2']}\ (\beta'')$$

Fig. 7. The parallel β-relation for λ^{var}

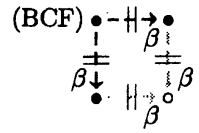
4 Confluence

The previous section establishes a rather large fragment of the λ^{var}-calculus as susceptible to primitive equational reasoning. This section summarises and elaborates on our formally verified efforts to bring this to bearing on β-confluence [25]. We also present proofs that apply the methodology to prove η- and $\beta\eta$-confluence.

4.1 β-Confluence

The $\dashrightarrow_\beta$-relation does not enjoy the diamond property because a redex that is contracted in one direction of a divergence can be duplicated (and erased) in the other direction by the substitution operator. As shown by Tait and Martin-Löf, the potential divergence "blow-up" does not materialise because it can be controlled by *parallel* reduction. Please refer to Figure 7 for the λ^{var}-version of this relation.

Lemma 24



Proof Rather than prove this property by an exhaustive case-splitting, thus resulting in a *minimally* resolving end-term, Takahashi observed that the considered diamond can be diagonalised by the relation that contracts all redexes in one step, i.e., by a *maximally* resolving end-term [23]. As we saw in Section 3 this is within reach of the structural proof principles of λ^{var}. □

The Full Proof Burden A real version of the parallel β-relation on syntax can be defined along the lines of Definition 17 (which, further to Lemma 21, turns out to be the *real* real parallel β-relation).

Definition 25 $\lfloor e_1 \rfloor \twoheadrightarrow\!\!\!\!\!\!\rightarrow_\beta \lfloor e_2 \rfloor \Leftrightarrow^{def} e_1 \twoheadrightarrow_\alpha ; \twoheadrightarrow_\beta ; \twoheadrightarrow_\alpha e_2$

In order to prove the diamond property for $\twoheadrightarrow\!\!\!\!\!\!\rightarrow_\beta$, we need some measure of commutativity between α- and

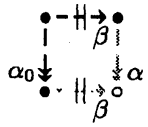
Fresh-Naming As the general α-/β-commutativity result is not provable, we introduce the following restricted α-relation, which only fresh-names.

Definition 26

$$e \dashrightarrow_{\alpha_0} e' \Leftrightarrow^{\mathrm{def}} \exists z.e \overset{z}{\dashrightarrow}_{i\alpha} e' \wedge z \notin \mathrm{FV}(e) \cup \mathrm{BV}(e)$$

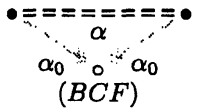
The fresh-naming α-relation can straightforwardly be proven to commute with the parallel (actually, any) β-relation with the proviso that the resolving α-steps are not necessarily fresh-naming (because of β-incurred term duplication).

Lemma 27



Similarly, the fresh-naming α-relation can be shown to resolve α-equivalence to a BCF (although the formal proof of this is surprisingly involved, cf. [25]).

Lemma 28



Applying Administration With these results in place, we can lift Lemma 24 to the real λ-calculus.

Lemma 29 $\Diamond(\twoheadrightarrow\!\!\!\!\!\!\rightarrow_\beta) \wedge \Diamond(\dashrightarrow\!\!\!\!\!\!\rightarrow_\alpha ; \twoheadrightarrow_\beta)$

Proof As for the left-most conjunct, see Figure 8 for the step by step resolution of the definitionally-given syntactic divergence. We trust the steps are self-evident and that it can be seen that a slight adaptation of the figure also proves the right-most conjunct. □

We are now in a position to establish β-confluence.

Theorem 30

$$\begin{aligned}&\mathrm{Confl}(\longrightarrow_\beta) \wedge \mathrm{Confl}(\dashrightarrow_{\alpha\beta})\\ &\qquad \wedge \mathrm{Confl}(\dashrightarrow_{\alpha^{\mathrm{Cu}}\beta^{\mathrm{Cu}}})\\ &\qquad \wedge \mathrm{Confl}(\longrightarrow_{\beta^{\mathrm{Hi}}})\end{aligned}$$

Proof The two top-most conjuncts are equivalent by Lemma 18. They can also be proved independently by applying the Diamond Tiling Lemma of Appendix C to the corresponding conjunct in Lemma 29. The third conjunct follows by Lemmas 15 and 16. The final conjunct follows in an analogous manner. □

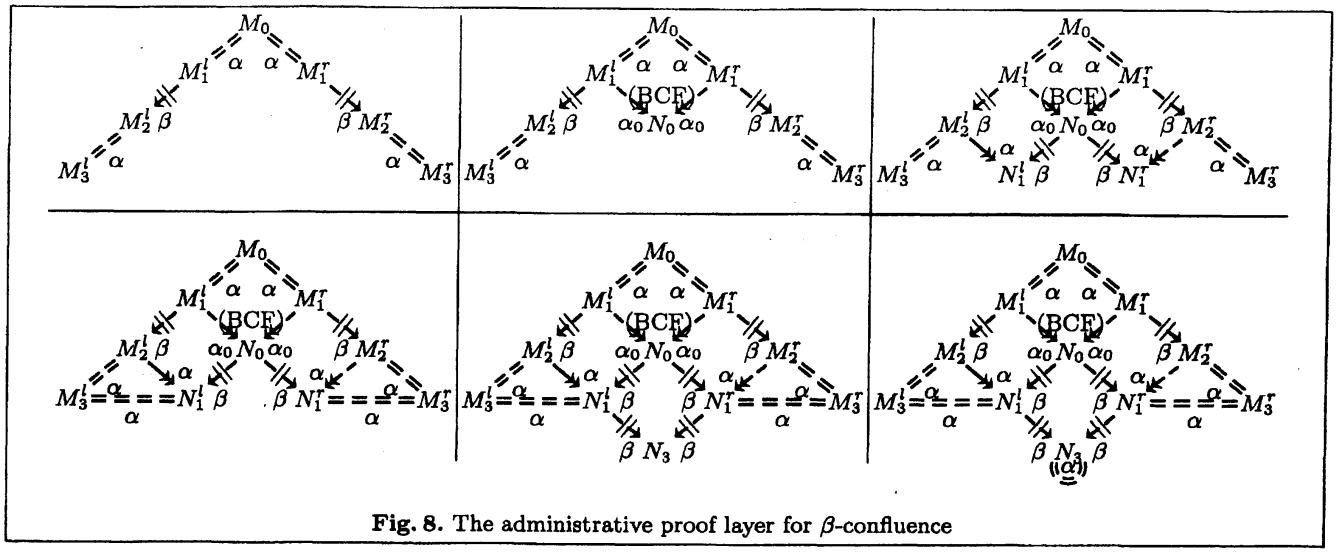


Fig. 8. The administrative proof layer for β-confluence

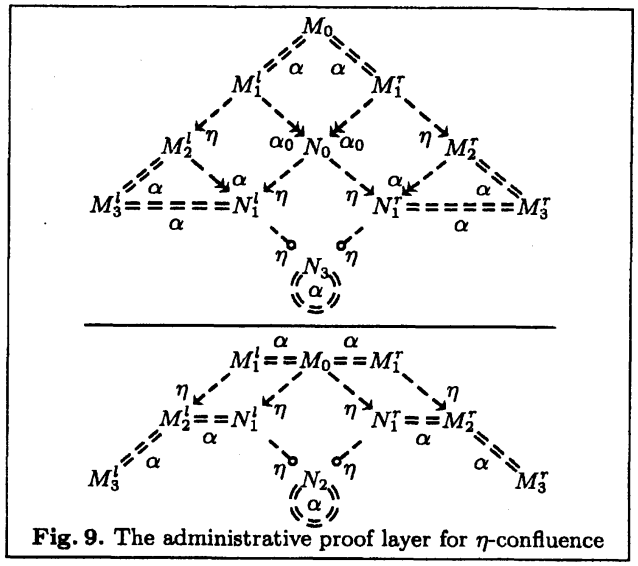
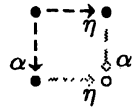


Fig. 9. The administrative proof layer for η-confluence

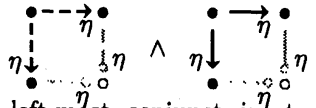
4.2 η-Confluence

Unlike the β-relation, η-reduction is natively renaming-free:

Lemma 31 (α/η **Commutativity**)



Lemma 32 (η **Commutativity**)



Proof The left-most conjunct is straightforwardly provable by structural means. The proof of the right-most property follows from the left-most as displayed in Figure 9. The top part of the figure is a proof by the *general* method; the lower part is an optimised version that takes advantage of η commuting with α, not just with α_0. □

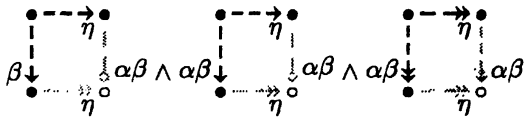
Theorem 33 $\mathrm{Confl}(\dashrightarrow_\eta) \wedge \mathrm{Confl}(\longrightarrow_\eta) \wedge \mathrm{Confl}(\dashrightarrow_{\alpha\eta})$

Proof The two left-most conjuncts can be established from the corresponding conjuncts in Lemma 32 by the Hindley-Rosen Lemma of Appendix C. The right-most conjunct can be established either by the Commuting Confluence Lemma of Appendix C applied to the left-most conjunct and generalisations of Lemmas 11 and 31 or, alternatively, it can be observed that the two right-most conjuncts are equivalent by Lemma 18. □

4.3 $\beta\eta$-Confluence

Since the η-relation is natively renaming-free and the β-relation relies on the α-relation, we must show that η-commutes with combined $\alpha\beta$-reduction in order to apply the Commuting Confluence Lemma of Appendix C.

Lemma 34



Proof The proof of the left-most conjunct is straightforward. The α-step in the resolution on the right is needed for the obvious divergence on $\lambda x.(\lambda y.e)x$, with $x \neq y$. The middle conjunct combines the left-most conjunct and Lemma 31. The right-most conjunct follows from the middle by the Hindley-Rosen Lemma of Appendix C. □

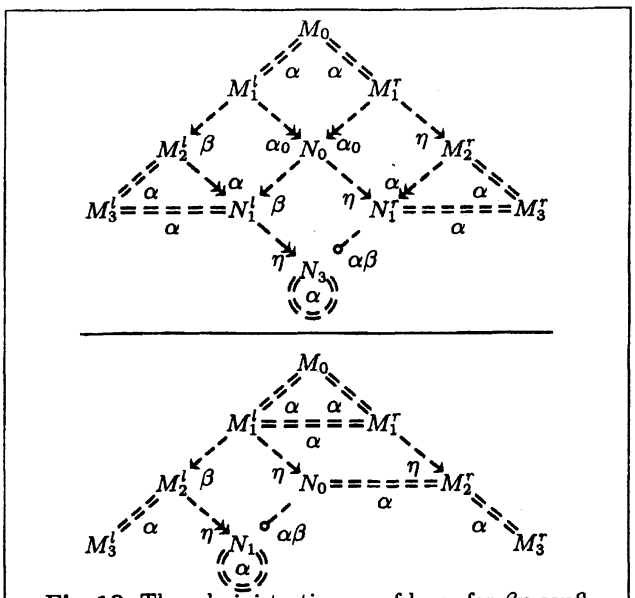
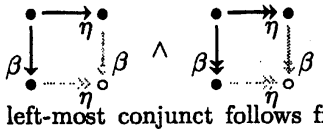


Fig. 10. The administrative proof layer for $\beta\eta$-confl

Lemma 35



Proof The left-most conjunct follows from the left-most conjunct of Lemma 34 as shown in Figure 10. The top part of the figure is by the *general* method; the lower part is an optimisation based on (full) $\alpha\eta$-commutativity, Lemma 31. The right-most conjunct follows by the Hindley-Rosen Lemma of Appendix C. □

Theorem 36 $\mathrm{Confl}(\longrightarrow_{\beta\eta}) \wedge \mathrm{Confl}(\dashrightarrow_{\alpha\beta\eta})$

Proof We first observe that the two conjuncts are equivalent by Lemma 18. They can also be proved independently by the Commuting Confluence Lemma of Appendix C applied to Theorems 30 and 33 as well as Lemma 35 and Lemma 34, respectively. □

5 Residual β-Confluence

We say that the reflexive, transitive closure of a residual relation is the associated *development* relation, a step of which is said to be *complete* if the target term does not contain a mark, unMarked(−). With this terminology in place, we define a weakened version of the strong finite development property.[6]

[6] The strong finite development property also requires that the residual relation is strongly normalising. It is typically used to prove (residual) confluence.

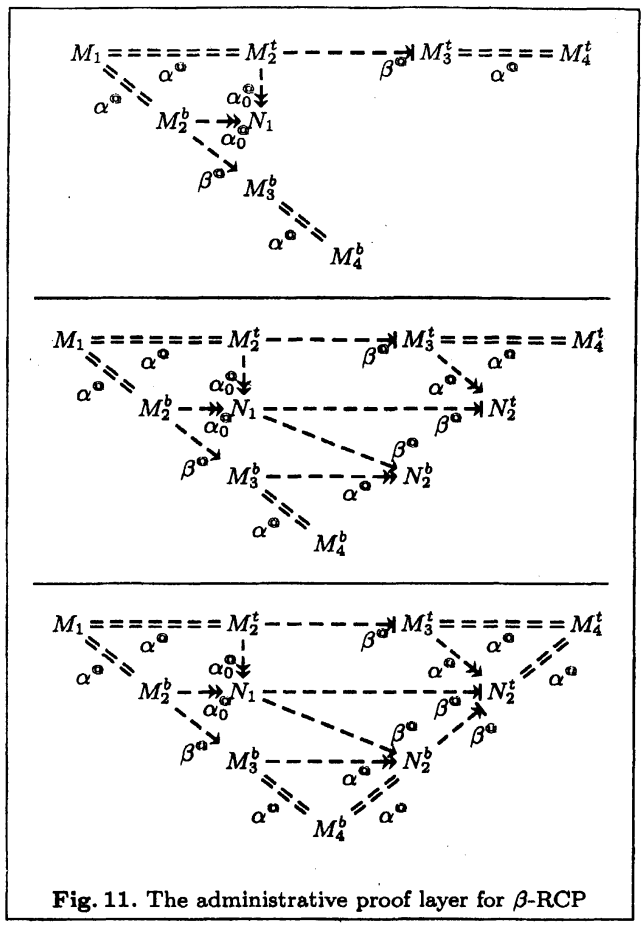


Fig. 11. The administrative proof layer for β-RCP

Definition 37 *Let* $\longrightarrow_{@}$ *be the residual relation of* $\longrightarrow$. *We say that* $\longrightarrow$ *enjoys the* strong weakly-finite development property, SWFDP($\longrightarrow$), *if*

1. $t \longrightarrow_{@} t' \Rightarrow \exists t''.t' \longrightarrow_{@} t'' \wedge \mathrm{unMarked}(t'')$
   — *developments can be completed*
2. $t \longrightarrow_{@} t_i \wedge \mathrm{unMarked}(t_i) \wedge i \in \{1,2\} \Rightarrow t_1 = t_2$
   — *completions are unique*

To motivate the name of the property, we see that, indeed:

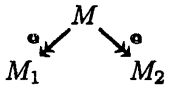
Proposition 38 $\mathrm{SWFDP}(\longrightarrow) \Rightarrow \mathrm{WN}(\longrightarrow_{@})$[7]

Proof By Definition 37, 1. and reflexivity of $\longrightarrow_{@}$. □

Surprisingly, perhaps, we have that already the SWFDP implies residual confluence.

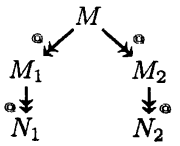
Lemma 39 $\mathrm{SWFDP}(\longrightarrow) \Rightarrow \mathrm{Confl}(\longrightarrow_{@})$

Proof Consider the following divergence:



[7] The predicate WN(−) stands for Weak Normalisation and means that all terms reduce to a normal form.

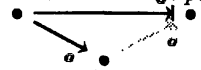
By Definition 37, 1., there exist N_1, N_2, such that unMarked(N_1), unMarked(N_2) and:



By transitivity of $\longrightarrow\!\!\!\!\!\rightarrow_{@}$ and Definition 37, 2., we see that, in fact, $N_1 = N_2$ and we are done. □

With direct reference to Section 3, we define the following property, which is fairly easily proven to be equivalent to the SWFDP.

Definition 40 *A relation, $\longrightarrow$, enjoys the residual-completion property,* RCP($\longrightarrow$), *if there exists a residual-completion relation, $\longrightarrow\!\!\!\!\!\rightarrow_{@}$, such that:*

1. $\longrightarrow\!\!\!\!\!\rightarrow_{@} \subseteq \longrightarrow\!\!\!\!\!\!\rightarrow_{@}$
   — *residual-completion is a development*
2. 
   — *residual-completion totally completes*
3. 
   — *residual-completion is residually co-final*

Lemma 41 RCP($\longrightarrow$) $\Leftrightarrow$ SWFDP($\longrightarrow$)

Our interest in the RCP is its constructive nature, in particular when the residual-completion relation is defined as a computable function the way we did in Section 3.

Lemma 42 RCP($\longrightarrow_\beta$) $\wedge$ SWFDP($\longrightarrow_\beta$)

Proof We prove the left-most conjunct. Clause 1. follows from the easily established fact that $\dashrightarrow\!\!\!\!\!\rightarrow_{\beta@} \subseteq \dashrightarrow\!\!\!\!\!\!\rightarrow_{\beta@}$. Clause 2 follows from Lemmas 21 and 28. Finally, Clause 3 is proved as shown in Figure 11. □

Theorem 43 Confl($\longrightarrow_{\beta@}$) $\wedge$ Confl($\dashrightarrow_{\alpha@\beta@}$)

We see that SN($\longrightarrow_{\beta@}$) (i.e., the difference between the SWFDP and the strong finite development property) is not needed for concluding confluence from a residual analysis of the β-relation, something which is in stark contrast to established opinion [2, p.283]. Strong finite development essentially implies confluence through Newman's Lemma, thus relying crucially on the (non-equational) SN-property for the residual relation. We think it a nice "purification" of the equational import of residual theory that an externally justified termination property is not needed for concluding confluence.

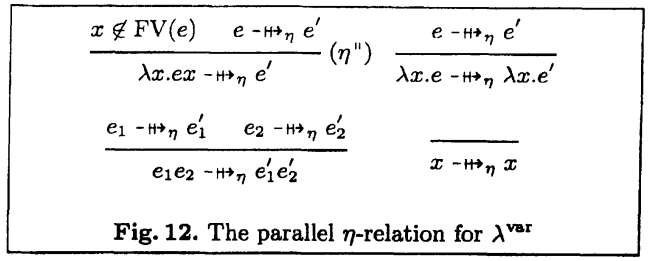


Fig. 12. The parallel η-relation for λ^{var}

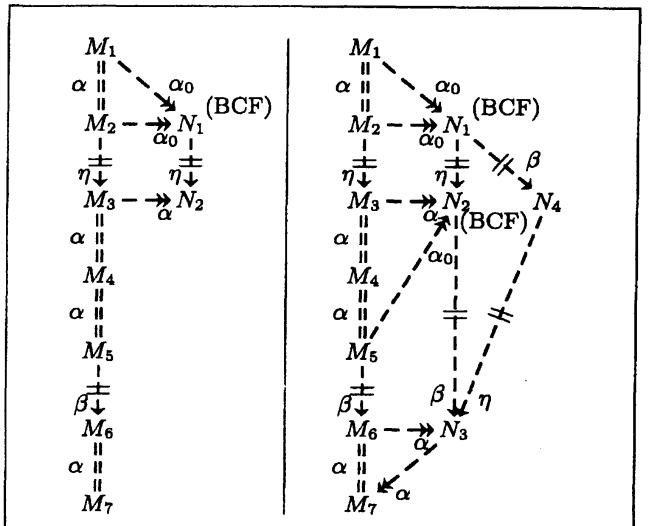


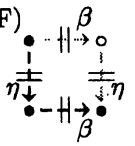
Fig. 13. The administrative proof layer for η-postponing

6 η-over-β-Postponement

As well as condensing Tait and Martin-Löf's use of parallel β-reduction for proving β-confluence, Takahashi [23] also shows how to adapt the parallel-reduction technology to other typical situations in the equational theory of the λ-calculus. One such situation is for proving η-over-β-postponement, cf. Figure 12. The proof presented by Takahashi [23] essentially goes through up to BCF-initiality as it stands, albeit not completely. Rather than focusing on the low-level technical details, this section merely shows the Administrative and Abstract proof layers of our formalisation of Takahashi's proof.

The notion of commutativity that we have considered so far is orthogonal in nature to that employed in the η-over-β-Postponement Theorem. Whereas the former can be described as *divergence commutativity*, this section focuses on *composition commutativity*.

Lemma 44 (BCF)



Proof The parallel η-relation is used to allow for the duplication of a η-redex by the β-contraction when the latter is performed first. The parallel β-relation, on the other hand, is used. e.g., for the following situation:

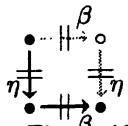
$$(\lambda x.(\lambda y.e_1)x)e_2 \dashrightarrow_\eta (\lambda y.e_1)e_2 \dashrightarrow_\beta e_1[y := e_2]$$

This reduction sequence commutes into a leading parallel β-step with a trailing η-step, which is in this case is reflexive:

$$(\lambda x.(\lambda y.e_1)x)e_2 \twoheadrightarrow_\beta e_1[y := x][x := e_2]$$

BCF-initiality is used to enable the double (n-fold, in general) substitution in the commuted reduction sequence. □

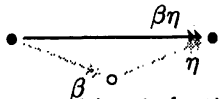
Lemma 45



Proof Please refer to Figure 13 for the details of the proof. A novel aspect of the proof is the existence of an α_0-step from M_5 to N_2. By construction, we know that the two terms are α-equivalent. A simple lemma shows that N_2 is a BCF because η-reduction preserves BCFs. The final result that is needed, i.e., that α_0-reduction can reach any BCF that is α-equivalent to the start term, can also be proved by structural means but it is not as straightforward as could be imagined. This is due to the need for the target BCF to be *any* BCF. □

With the one necessary technical lemma in place, we present the postponement theorem.

Theorem 46



Proof By reflexive, transitive induction in $\longrightarrow\!\!\!\!\!\rightarrow_{\beta\eta}$. The only interesting case is the transitive case, which follows in a manner akin to the Hindley-Rosen Lemma of Appendix C using Lemma 45. □

7 β-Standardisation

Standardisation is also a composition-commutativity result like postponement. It is a very powerful result that, informally speaking, says that any reduction sequence can be performed left-to-right. Standardisation implies results such as the left-most reduction lemma, etc., [2], and guarantees the existence of evaluation-order independent semantics [20].

This section addresses three different approaches to proving standardisation due to Mitschke [18], Plotkin [20], and David [5], respectively. The three approaches are fairy closely related, with Plotkin's proof bridging the other two, so to speak. Mitschke's and Plotkin's proofs both use semi-standardisation while David's and Plotkin's both can be described as *absorption* standardisation. In spite (actually *because*) of this, only Plotkin's approach is formally provable by the proof principles we are considering. We shall examine the failures of the other two proofs closely.

$$\frac{\mathrm{Capt}_x(e_1) \cap \mathrm{FV}(e_2) = \emptyset}{(\lambda x.e_1)e_2 \dashrightarrow_{\beta^{\mathrm{wh}}} e_1[x := e_2]}\,(\beta^{\mathrm{wh}}) \qquad \frac{e_1 \dashrightarrow_{\beta^{\mathrm{wh}}} e_1'}{e_1e_2 \dashrightarrow_{\beta^{\mathrm{wh}}} e_1'e_2}\,(@^{\mathrm{wh}})$$

Fig. 14. Weak-head β-reduction

$$\frac{e_1 \dashrightarrow_{\beta^{\mathrm{I}}} e_1'}{e_1e_2 \dashrightarrow_{\beta^{\mathrm{I}}} e_1'e_2}\,(@_1^{\mathrm{I}}) \qquad \frac{e_2 \dashrightarrow_\beta e_2'}{e_1e_2 \dashrightarrow_{\beta^{\mathrm{I}}} e_1e_2'}\,(@_2^{\mathrm{I}})$$

$$\frac{e \dashrightarrow_\beta e'}{\lambda x.e \dashrightarrow_{\beta^{\mathrm{I}}} \lambda x.e'}\,(\lambda^{\mathrm{I}})$$

$$\frac{}{x \twoheadrightarrow_{\beta^{\mathrm{I}}} x}\,(\mathrm{Var}_{\shortparallel}^{\mathrm{I}}) \qquad \frac{e_1 \twoheadrightarrow_{\beta^{\mathrm{I}}} e_1' \quad e_2 \twoheadrightarrow_\beta e_2'}{e_1e_2 \twoheadrightarrow_{\beta^{\mathrm{I}}} e_1'e_2'}\,(@_{\shortparallel}^{\mathrm{I}})$$

$$\frac{e \twoheadrightarrow_\beta e'}{\lambda x.e \twoheadrightarrow_{\beta^{\mathrm{I}}} \lambda x.e'}\,(\lambda_{\shortparallel}^{\mathrm{I}})$$

Fig. 15. Inner and parallel inner β-reduction

7.1 Semi-Standardisation with Hereditary Recursion

In this section, we shall pursue a slight adaptation of Takahashi's adaptation [23] of Mitschke's proof [18]. Instead of head and a corresponding notion of inner reduction, we base the proof on weak-head reduction. This does not affect the formal status of the proof technique but does allow us to reuse the results of this section when pursuing Plotkin's approach. The main proof burden is to show that (weak-)head redexes can contracted before any inner redexes, so-called semi-standardisation.

Definition 47 *Weak-head β-reduction, $\dashrightarrow_{\beta^{\mathrm{wh}}}$, is defined in Figure 14. The corresponding (strong) inner, $\dashrightarrow_{\beta^{\mathrm{I}}}$, and parallel inner, $\twoheadrightarrow_{\beta^{\mathrm{I}}}$, β-relations are defined in Figure 15.*

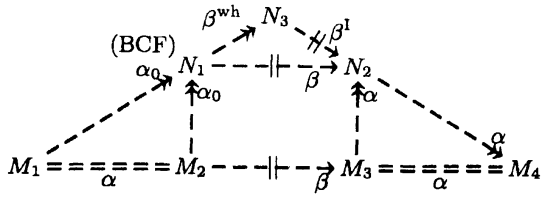


Fig. 16. The administrative proof layer for weak-head before inner parallel decomposition

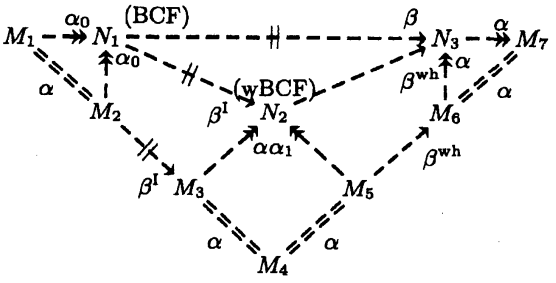
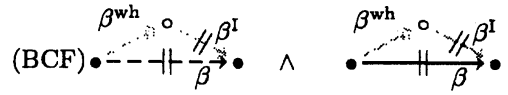


Fig. 17. The administrative proof layer for parallelization of weak-head after inner

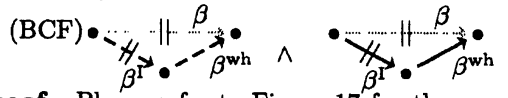
Lemma 48



Proof Please refer to Figure 16 for the proof of the right-most conjunct based on the left-most conjunct, wich, in turn, is proved by rule induction in $\text{-H}\twoheadrightarrow_\beta$. □

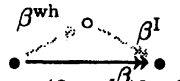
The use of BCF-initiality in the left-most conjunct above guarantees that weak-head redexes can be contracted without waiting for the contraction of an inner redex to eliminate a variable clash.

Lemma 49



Proof Please refer to Figure 17 for the proof of the right-most conjunct based on the left-most. We first note that the figure invokes the obvious adaptation of Lemma 27 to $\text{-H}\twoheadrightarrow_{\beta^I}$. Although the proof as a whole is similar to that of η-over-β-postponement, cf. Lemma 45, we do not have that $\text{-H}\twoheadrightarrow_{\beta^I}$ preserves BCFs, as is the case with $\text{-H}\twoheadrightarrow_\eta$. Instead, we can introduce a weakened notion of BCF, wBCF, that allows identical binders to occur in adjacent positions (but not nested and not coinciding with any free variables) and show that $\text{-H}\twoheadrightarrow_{\beta^I}$ sends BCFs to wBCFs. In the same manner that α_0-reduction and the BCF-predicate correspond to each other, we can introduce an α_1-relation that corresponds to the wBCF-predicate. The α_1-relation is less well-behaved than the α_0-relation but we can, at least, show that it commutes with $\dashrightarrow_\beta$ (and thus $\dashrightarrow_{\beta^{wh}}$), up to α-resolution. The left-most conjunct of the lemma, follows by rule induction in $\text{-H}\twoheadrightarrow_{\beta^I}$. □

Lemma 50 (Semi-Standardisation)



Proof From Lemmas 48 and 49, cf. the obvious reflexive, transitive generalisation of Lemma 53 in the transitive case. □

At this point, the idea is to recursive over the $\circ$ in Lemma 50 and show that the sub-terms in which the outgoing $\twoheadrightarrow_{\beta^I}$-step are ordinary β-steps, themselves can be semi-standardised and so on. Unfortunately, the $\circ$ is quantified over α-equivalence classes, for which no recursion is possible and we are stuck.

7.2 Hereditary Weak-Head Standardisation

Plotkin [20] defines standardisation as the least contextually-closed relation on terms that enjoys left-absorptivity over weak-head reduction. The following presentation of the proof methodology owes a great debt to McKinna and Pollack [17]. The difference between their and our presentation is that we focus on provability with structural induction, etc., while they work with an alternative syntactic framework that is *derived* from first-order abstract syntax with *two*-sorted variable names. The proof requirements in their setting and in ours are substantially different as a result.

A First (Failing) Approach A first approach, which immediately fails, is to define Plotkin's relation directly on terms.

$$\frac{e \dashrightarrow_{\beta^{wh}} e' \quad e' \succ\!\!\dashrightarrow_{\mathrm{P}} e''}{e \succ\!\!\dashrightarrow_{\mathrm{P}} e''} \qquad \frac{}{x \succ\!\!\dashrightarrow_{\mathrm{P}} x}$$

$$\frac{e_1 \succ\!\!\dashrightarrow_{\mathrm{P}} e_1' \quad e_2 \succ\!\!\dashrightarrow_{\mathrm{P}} e_2'}{e_1 e_2 \succ\!\!\dashrightarrow_{\mathrm{P}} e_1' e_2'} \qquad \frac{e \succ\!\!\dashrightarrow_{\mathrm{P}} e'}{\lambda x.e \succ\!\!\dashrightarrow_{\mathrm{P}} \lambda x.e'}$$

As standardisation pertains to *all* β-reductions (i.e., $\twoheadrightarrow_\beta$, not just $\dashrightarrow_\beta$), the naive approach needs the full λ-calculus to be renaming-free, which it is not. The problem manifests itself in the required administrative proof layer for the standardisation property and its exact nature is of independent interest. The point is that, even if it is possible to prove the following key property (which, in fact, seems to be the case[8]), we cannot prove

[8] Coincidentally, it is interesting to note that the proof of the property can only be conducted by rule induction in $\succ\!\!\dashrightarrow_{\mathrm{P}}$ and not in $\dashrightarrow_\beta$.

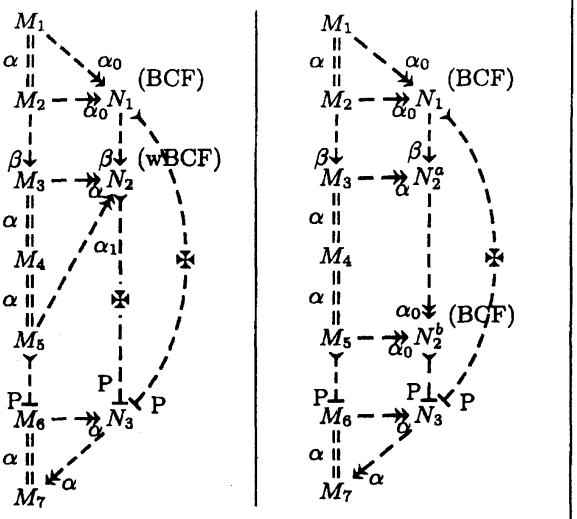


Fig. 18. Failed administrative proof layer for left-absorptivity of progression standardisation

full standardisation but at most standardisation of the renaming-free fragment of the λ^{var}-calculus.

Please refer to Figure 18 for the only two sensible approaches to the administrative proof layer for the following property, which is derived from the one above.

Non-Lemma 51

The left-most diagram in the figure attempts to align itself with Figure 13, which fails because $\succ\!\!\dashv_{\mathrm{P}}$ only commutes with $\dashrightarrow_{\alpha_0}$. The right-most diagram adheres to this and fails because of the inserted $\dashrightarrow\!\!\!\!\rightarrow_{\alpha_0}$, which we cannot incorporate into the syntactic version of the property. It is even straightforward to come up with a counter-example.

$$(\lambda s.ss)(\lambda x.\lambda y.xy) \dashrightarrow_\beta (\lambda x.\lambda y.xy)(\lambda x.\lambda y.xy)$$

We can turn the end-term into an α-equivalent BCF, as it happens, which standardises:

$$(\lambda x_1.\lambda y_1.x_1y_1)(\lambda x_2.\lambda y_2.x_2y_2) \succ\!\!\dashv_{\mathrm{P}} \lambda y_1.\lambda y_2.y_1y_2$$

As the end-term of this step uses the two y copies nested within each other, we see that the original start term does not standardise to it.

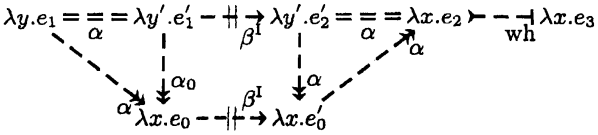


Fig. 19. The administrative proof layer for the $(\lambda_{\succ\dashv_{\mathrm{wh}}})$-case of Lemma 54

Combining Term Structure and α-Collapsed Reduction In order to avoid these problems, we adapt the above definition slightly.

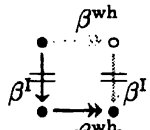
Definition 52

$$\frac{\lfloor e\rfloor \longrightarrow\!\!\!\!\!\rightarrow_{\beta^{\mathrm{wh}}} \lfloor e'\rfloor \quad e' \succ\!\!\dashv_{\mathrm{wh}} e''}{e \succ\!\!\dashv_{\mathrm{wh}} e''}\,(\mathrm{wh}_{\mathrm{pre}}) \qquad \frac{}{x \succ\!\!\dashv_{\mathrm{wh}} x}\,(\mathrm{V}_{\succ\dashv_{\mathrm{wh}}})$$

$$\frac{e_1 \succ\!\!\dashv_{\mathrm{wh}} e_1' \quad e_2 \succ\!\!\dashv_{\mathrm{wh}} e_2'}{e_1e_2 \succ\!\!\dashv_{\mathrm{wh}} e_1'e_2'}\,(@_{\succ\dashv_{\mathrm{wh}}}) \qquad \frac{e \succ\!\!\dashv_{\mathrm{wh}} e'}{\lambda x.e \succ\!\!\dashv_{\mathrm{wh}} \lambda x.e'}\,(\lambda_{\succ\dashv_{\mathrm{wh}}})$$

The definition mixes the advantages of being able to define relations inductively over terms with the use of reduction in the real λ-calculus to avoid issues of renaming. Note, however, that, further to the failed proof of Lemma 4, it is by no means obvious whether this mixture will lend itself to primitive structural reasoning. The proof-technical issue surfaces in the $(\mathrm{V}_{\succ\dashv_{\mathrm{wh}}})$-case of the proof of Lemma 54.

Lemma 53



Proof The property can be derived from Lemmas 48 and 49 based on a suitable adaptation of the Hindley-Rosen Lemma, cf. Appendix C. □

The key technical lemma in the present standardisation proof development is the following *absorption* property.

Lemma 54

$$\lfloor e_1\rfloor \twoheadrightarrow\!\!\!\!\!\rightarrow_{\beta^{\mathrm{I}}} \lfloor e_2\rfloor \wedge e_2 \succ\!\!\dashv_{\mathrm{wh}} e_3 \Rightarrow e_1 \succ\!\!\dashv_{\mathrm{wh}} e_3$$

Proof The proof is by rule induction in $\succ\!\!\dashv_{\mathrm{wh}}$ and uses Lemma 53 before applying the I.H. and the definitional left-absorptivity over weak-head reduction when needed. As far as administration is concerned, the only interesting case is for abstraction.

Case $(\lambda_{\succ\dashv_{\mathrm{wh}}})$: We are considering the following situation (although this takes some effort to substantiate).

$$\lambda y.e_1 =\!\!=_\alpha \lambda y'.e_1' \twoheadrightarrow\!\!\!\!\!\rightarrow_{\beta^{\mathrm{I}}} \lambda y'.e_2' =\!\!=_\alpha \lambda x.e_2 \succ\!\!\dashv_{\mathrm{wh}} \lambda x.e_3$$

By Definition 47 and the case, we have $e_1' \twoheadrightarrow_\beta e_2'$ and $e_2 \succ\!\!\dashv_{\mathrm{wh}} e_3$. If $y' = x$, we can prove $e_2' =\!=_\alpha e_2$, which means that we are considering $\lfloor e_1' \rfloor \twoheadrightarrow_\beta \lfloor e_2' \rfloor \succ\!\!\dashv_{\mathrm{wh}} e_3$, so to speak. From Lemma 48, we thus have: $\lfloor e_1' \rfloor \longrightarrow\!\!\!\!\rightarrow_{\beta^{\mathrm{wh}}} \lfloor e_1'' \rfloor \twoheadrightarrow_{\beta^{\mathrm{I}}} \lfloor e_2' \rfloor \succ\!\!\dashv_{\mathrm{wh}} e_3$. An application of the I.H. and an invocation of the $(\mathrm{wh}_{\mathrm{pre}})$-rule will then give us that $e_1' \succ\!\!\dashv_{\mathrm{wh}} e_3$ and we have $\lambda x.e_1' \succ\!\!\dashv_{\mathrm{wh}} \lambda x.e_3$ by the $(\lambda_{\succ\!\!\dashv_{\mathrm{wh}}})$-rule. A final (reflexive) application of the $(\mathrm{wh}_{\mathrm{pre}})$-rule thus finishes the case: $\lambda y.e_1 \succ\!\!\dashv_{\mathrm{wh}} \lambda x.e_3$. Unfortunately, we can not guarantee $y' = x$. Instead, Figure 19 shows how to overcome this using our general administrative proof-layer technology, cf. Figure 3. Based on the upper line, we first rewrite $\lambda y'.e_1'$ to (the BCF) $\lambda x.e_0$ (although it takes some effort to substantiate that this is possible). The commuting square involving $\lambda x.e_0'$ can then be constructed by the obvious adaptation of Lemma 27 and the diagram can finally be closed based on Lemma 11. To show that $\lambda y.e_1 \succ\!\!\dashv_{\mathrm{wh}}$-standardises to $\lambda x.e_3$, first apply the reasoning above to show that $\lambda x.e_0$ does and, then, use the $(\mathrm{wh}_{\mathrm{pre}})$-rule reflexively to show the result we are after.

Other Cases: Fairly straightforward. □

Theorem 55 $\lfloor e_1 \rfloor \longrightarrow\!\!\!\!\rightarrow_\beta \lfloor e_2 \rfloor \Rightarrow e_1 \succ\!\!\dashv_{\mathrm{wh}} e_2$

Proof By reflexive, left-transitive induction in $\longrightarrow\!\!\!\!\rightarrow_\beta$. The reflexive case is a straightforward structural induction. The left-transitive case follows by an I.H.-application followed by a case-split on the considered $\longrightarrow_\beta$-step into $\longrightarrow_{\beta^{\mathrm{wh}}}$ and $\longrightarrow_{\beta^{\mathrm{I}}}$ (seeing that we can show that the union of the latter two is the former). In case of $\longrightarrow_{\beta^{\mathrm{wh}}}$, we are done by definition of $\succ\!\!\dashv_{\mathrm{wh}}$. In case of $\longrightarrow_{\beta^{\mathrm{I}}}$, we are done by Lemma 54. □

7.3 (Failing) Progression Standardisation

An alternative proof development for standardisation was proposed by David [5] and pursued, more or less independently, in [13–15]. The idea is to define a standardisation relation directly by induction over terms (although this is only done implicitly in [5]): $\succ\!\!\dashv_{\mathrm{prg}}$, and to show that this relation right-absorbs the ordinary β-relation. In that sense, the proof development is the dual approach to what we considered in the previous section. Informally, the key technical point is to contract terms as follows, cf. [13, 15]:[9]

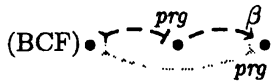
$$\frac{(..(e[x := e_0]e_1)..)e_k \succ\!\!\dashv_{\mathrm{prg}} e'}{(..((\lambda x.e)e_0)e_1..)e_k \succ\!\!\dashv_{\mathrm{prg}} e'}$$

[9] In order for the relation to make sense in the current setting, it is necessary to supply it with a finite axiomatisation, which can be done.

This ensures that contraction *progresses* from left-to-right while at the same time allowing newly created redexes to be contracted. Other rules allow redexes not to be contracted as the relation otherwise would be left-most reduction.

Right-Absorptivity As mentioned, the key technical lemma is purported to show right-absorptivity of $\succ\!\!\dashv_{\mathrm{prg}}$ over $\dashrightarrow_\beta$, which appears to be straightforward, at least in the case of the above contraction rule [5, 13–15].

Non-Lemma 56



Unfortunately, not even the BCF-initial version of the property is true. The following is a counter-example.

$$(\lambda s.ss)(\lambda x.(\lambda y.xy)z) \succ\!\!\dashv_{\mathrm{prg}} (\lambda y.(\lambda x.xz)y)z \dashrightarrow_\beta (\lambda y.yz)z$$

The problem in the counter-example is the last step of the standardisation, which amounts to the contraction of the redex involving the inner y-abstraction below.

$$(\lambda y.(\lambda x.(\lambda y.xy)z)y)z$$

As it happens, this is the point where the considered $\dashrightarrow_\beta$-step (i.e., the contraction of the redex involving the x-abstraction) must be inserted but that is not possible because of a clash with the inner y-abstraction.

Left-Absorptivity In sharp contrast with the above (and surprisingly, at first), it turns out that it is possible to prove left-absorptivity, as also seen at the beginning of Section 7.2.

The difference between right- and left-absorptivity is that the universal quantification over $\succ\!\!\dashv_{\mathrm{prg}}$ covers far fewer steps in the latter case than in the former. As we saw, this manifests itself when trying to prove standardisation for the real λ-calculus.

Non-Lemma 57

The counter-example at the beginning of Section 7.2 applies.

8 Conclusion

Standard, informal practice in the programming language theory community when using structural induction and related proof principles is to assume that variables clashes are not an issue (aka Barendregt's Variable Convention). We have shown this to be formally correct for a range of standard properties, possibly up to BCF-initiality, cf. Lemmas 21, 22, 24, 32, 34, 44, 48, 49, and 54. For the most part, we have been able to show that the undertaken proof burden resolution is formally incomplete in the sense that the formal proof burden can be met by the addition of a fairly simple administrative proof layer, cf. Figures 8, 9, 10, 11, 13, 16, and 17. The administrative proof layers mostly rely on the same additional lemmas, thus preventing a blow-up of proof obligations. We studied standardisation in some detail and found that only one out of three proof techniques appears to be amenable to the use of structural induction, etc..

A Commutative Diagrams

We use commutative diagrams in three different ways, which are distinguished in the way they write vertices.

A.1 Vertices as Terms

When written with terms as vertices, commutative diagrams simply describe reduction scenarios.

A.2 Vertices as M's, N's

We shall see next that commutative diagrams are used to express rewriting predicates such as:

> "For all terms, such that, ..., there exist terms, such that, ..."

In order to prove these results, we start by writing M's for the universally quantified terms and gradually introduce N's from supporting lemmas to eventually substantiate the existence claims. Please note that we use ✠ to signify "claimed" existences that are impossible.

A.3 Vertices as •'s, ∘'s

Formally, a commutative diagram of this nature is a set of vertices and a set of directed edges between pairs of vertices. Informally, the colour of a vertex (• vs ∘) denotes quantification modes over terms, universal and existential, respectively. A vertex may be guarded by a predicate. Edges are written as the relational symbol they pertain to and are either full-coloured (black) or half-coloured (gray). Informally, the colour indicates assumed and concluded relations, respectively. An edge connected to a ∘ must be half-coloured. A diagram must be type-correct on domains. A property is read off of a diagram thus:

1. write universal quantifications for all •'s
2. assume the full-coloured relations and the validation of any guard for a •
3. conclude the guarded existence of all ∘s and their relations

The following diagram and property are thus equivalent.

$$\begin{array}{ll} (P)\bullet \longrightarrow \bullet & e_1 \to e_2 \ \wedge\ e_1 \to e_3 \ \wedge\ P(e_1) \\ \downarrow \quad \downarrow & \Downarrow \\ \bullet \cdots\!\!\rightarrow \circ\,(Q) & \exists e_4 \,.\, e_2 \to e_4 \ \wedge\ e_3 \to e_4 \ \wedge\ Q(e_4) \end{array}$$

B Notation and Terminology

We say that a term *reduces* to another if the two are related by a *reduction* relation and we denote the relationship by an infix arrow between the two terms. The "direction" of the reduction should be thought of as being from-left-to-right. The sub-term of the left-hand side that a reduction step "acts upon" is called the *redex* of the reduction and it is said to be *contracted*.

- The converse of a relation, $\to$, is written $(\to)^{-1}$.
- Composition is:

$$a \to_1; \to_2 c \Leftrightarrow^{\text{def}} \exists b \,.\, a \to_1 b \ \wedge\ b \to_2 c$$

- Given two reduction relations $\to_1$ and $\to_2$, we have: $\to_{1,2} \ =^{\text{def}} \ \to_1 \cup \to_2$. If no confusion is possible, we omit the comma.
- The reflexive closure of a relation is:[10]

$$\frac{e_1 \to e_2}{e_1 \multimap e_2} \qquad \frac{}{e \multimap e}$$

- The reflexive, transitive closure is:

$$\frac{e_1 \to e_2}{e_1 \twoheadrightarrow e_2} \qquad \frac{}{e \twoheadrightarrow e} \qquad \frac{e_1 \twoheadrightarrow e_2 \quad e_2 \twoheadrightarrow e_3}{e_1 \twoheadrightarrow e_3}$$

  We will also denote $\twoheadrightarrow$ by $(\to)^\star$.
- The reflexive, transitive, and symmetric closure is:

$$\frac{e_1 \to e_2}{e_1 = e_2} \qquad \frac{}{e = e} \qquad \frac{e_1 = e_2 \quad e_2 = e_3}{e_1 = e_3} \qquad \frac{e_1 = e_2}{e_2 = e_1}$$

- The situation of a term reducing to two terms is called a *divergence*.

[10] This and the next two items are immediately associated with primitive induction principles. Equality, however, is only point-wise (or extensional), and no recursion principle is possible.

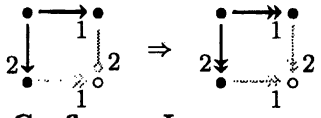
- Two diverging reductions, as defined above, are said to be *co-initial*.
- Dually, two reductions that share their end-term are said to be *co-final*.
- Co-initial reductions are *resolvable* if they compose with co-final reductions.
- A relation has the *diamond property*, $\diamond$, if any divergence can be resolved.
- A relation, $\rightarrow$, is *confluent*, Confl, if $\diamond(\twoheadrightarrow)$.

C Known Abstract Results

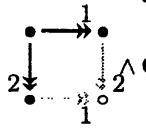
Diamond Tiling Lemma

$$(\exists \rightarrow_2 . \rightarrow_1 \subseteq \rightarrow_2 \subseteq \twoheadrightarrow_1 \wedge \diamond(\rightarrow_2)) \Rightarrow \diamond(\twoheadrightarrow_1)$$

Hindley-Rosen Lemma



Commuting Confluence Lemma



$\wedge\ \mathrm{Confl}(\rightarrow_1) \wedge \mathrm{Confl}(\rightarrow_2) \Rightarrow \mathrm{Confl}(\rightarrow_{1,2})$

References

1. Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland, Amsterdam, 1977.
2. Henk Barendregt. *The Lambda Calculus — Its Syntax and Semantics (Revised Edition)*. North-Holland, 1984.
3. Rod Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12, 1967.
4. H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.
5. René David. Une preuve simple de rèsultats classiques en λ calcul. *Comptes Rendus de l'Acadèmie des Sciences*, 320(11):1401–1406, 1995. Série I.
6. N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indag. Math.*, 34:381–392, 1972.
7. Joëlle Despeyroux and André Hirschowitz. Higher-order abstract syntax with induction in Coq. In Frank Pfenning, editor, *Proceedings of LPAR-5*, volume 822 of *LNAI*. Springer-Verlag, 1994.
8. Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In Philippe De Groote and J. Roger Hindley, editors, *Proceedings of TLCA-3*, volume 1210 of *LNCS*. Springer-Verlag, 1997.
9. Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In Longo [16], pages
10. Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In Longo [16], pages 214–224.
11. J. Roger Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle upon Tyne, 1964.
12. Martin Hofmann. Semantical analysis of higher-order abstract syntax. In Longo [16], pages 204–213.
13. Felix Joachimski and Ralph Matthes. Standardization and confluence for a lambda calculus with generalized applications. In Leo Bachmair, editor, *Proceedings of RTA-11*, volume 1833 of *LNCS*. Springer Verlag, 2000.
14. Ryo Kashima. On the standardization theorem for lambda-beta-eta-calculus. In *Proceedings of RPC'01*, 2001. Technical report, the Research Institute of Electrical Communication, Tohoku University, Japan.
15. Ralph Loader. Notes on simply typed lambda calculus. Technical report no. ECS-LFCS-98-381 of LFCS, University of Edinburgh, 1998.
16. Giuseppe Longo, editor. *Proceedings of LICS-14*. IEEE CS Press, 1999.
17. James McKinna and Randy Pollack. Some lambda calculus and type theory formalized. *Journal of Automated Reasoning*, 23(3–4), November 1999.
18. Gerd Mitschke. The standardization theorem for λ-calculus. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 25:29–31, 1979.
19. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 200X. To appear; a preliminary version appears in TACS-4 2001, LNCS 2215.
20. Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.
21. Carsten Schürmann. *Automating the Meta Theory of Deductive Systems*. PhD thesis, Carnegie Mellon University, 2000.
22. Allen Stoughton. Substitution revisited. *Theoretical Computer Science*, 59(3):317–325, August 1988.
23. Masako Takahashi. Parallel reductions in λ-calculus. *Information and Computation*, 118:120–127, 1995.
24. René Vestergaard and James Brotherston. The mechanisation of Barendregt-style equational proofs (the residual perspective). *Electronic Notes in Theoretical Computer Science*, 58(1), 2001. Invited version of MERLIN01 workshop paper.
25. René Vestergaard and James Brotherston. A formalised first-order confluence proof for the λ-calculus using one-sorted variable names. *Information and Computation*, 200X. To appear; special edition with selected papers from RTA01.